IMPROVING INTRUSION DETECTION IN UNIX-BASED NETWORKS

THESIS

David R. Landry
2LT, USAF

AFIT/GCS/ENG/94D-14

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/94D-14

IMPROVING INTRUSION DETECTION IN UNIX-BASED NETWORKS

THESIS
David R. Landry
2LT, USAF

AFIT/GCS/ENG/94D-14

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1994	Master's Thesis

4. TITLE AND SUBTITLE
IMPROVING INTRUSION DETECTION
IN UNIX-BASED NETWORKS

5. FUNDING NUMBERS

6. AUTHOR(S)
David R. Landry, 2nd Lieutenant, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCS/ENG/94D-14

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Computer security has not kept pace with the rapid growth of networked systems. Through its connection to the Internet, the Department of Defense is vulnerable to computer-based attacks. Current intrusion detection systems are still unproven, too complicated, or too costly for most system security officers to implement.

The attack methods used by system intruders are known and can be represented as groups of commands called attack signatures. This thesis investigates methods for detecting intruders by monitoring command usage. Testing was conducted in both controlled and uncontrolled circumstances.

With controlled testing, it was shown that 7 of the 11 signatures could be detected through command monitoring. Command recording deficiencies prevented all 11 signatures from being detected. With uncontrolled testing, users were monitored without their knowledge for one month. No actual attacks were observed, but there were 18 instances of false positives out of 145,066 monitored commands.

The implemented system was successful at detecting most attacks, with only a small percentage of false positives. This thesis is an intermediate step in exploring methods to better protect Air Force systems from attack. Future work should aim to detect attacks before they are fully completed by monitoring networks at the packet level.

14. SUBJECT TERMS	15. NUMBER OF PAGES
COMPUTER SECURITY; UNIX SECURITY; INTRUSION DETECTION	58
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. ***(If known)***

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

- **DOD** - See DoDD 5230.24, "Distribution Statements on Technical Documents."
- **DOE** - See authorities.
- **NASA** - See Handbook NHB 2200.2.
- **NTIS** - Leave blank.

Block 12b. Distribution Code.

- **DOD** - Leave blank.
- **DOE** - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
- **NASA** - Leave blank.
- **NTIS** - Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only)*.

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

★U.S.GPO:1990-0-273-271

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

AFIT/GCS/ENG/94D-14

IMPROVING INTRUSION DETECTION IN UNIX-BASED NETWORKS

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science (Computer Science)

David R. Landry, B.S.C.S.

2LT, USAF

December 1994

Table of Contents

List of Figures

List of Tables

Abstract

Computer security has not kept pace with the rapid growth of networked systems. Through its connection to the Internet, the Department of Defense is vulnerable to computer-based attacks. Current intrusion detection systems are still unproven, too complicated, or too costly for most system security officers to implement.

The attack methods used by system intruders are known and can be represented as groups of commands called attack signatures. This thesis investigates methods for detecting intruders by monitoring command usage. Testing was conducted in both controlled and uncontrolled circumstances.

With controlled testing, it was shown that 7 of the 11 signatures could be detected through command monitoring. Command recording deficiencies prevented all 11 signatures from being detected. With uncontrolled testing, users were monitored without their knowledge for one month. No actual attacks were observed, but there were 18 instances of false positives out of 145,066 monitored commands.

The implemented system was successful at detecting most attacks, with only a small percentage of false positives. This thesis is an intermediate step in exploring methods to better protect Air Force systems from attack. Future work should aim to detect attacks before they are fully completed by monitoring networks at the packet level.

IMPROVING INTRUSION DETECTION IN UNIX-BASED NETWORKS

I. Introduction

1.1 Background

The United States Air Force, along with other members of the Department of Defense (DOD), has always been interested in safeguarding its communications traffic. Much of this traffic today travels along the Internet, a connection of over 30,000 computer networks involving over 25 million people who live in 137 countries. Although classified information does not travel over the Internet, the U.S. Air Force is still concerned about unauthorized use of government computer equipment.

DOD computers around the world are being compromised on a weekly basis through their connections to Internet. This problem has become so serious that a former DOD official was quoted in *Federal Computer Week* as saying, "... on any given day DOD literally does not have control of five or six of its computer systems; the hackers do" (4). Figure 1.1 shows the U.S. military sub-networks which are known to have been attacked (4).

The unauthorized users are penetrating the system down to the root level. Root is the level at which a system administrator operates on a UNIX[1] system. Anyone acting as root has broad powers over the whole network such as the ability to read or write any file or assume the identity of any user on the network.

There are few security tools with which a system administrator can actively look for intruders (15, 23). In fact, most hackers are still discovered passively. A report presented at the 13th National Computer Security Conference in October 1990 noted, "Sixty percent of the sites that suffer intrusions learn about the problem only when other sites or the CERT/CC (Computer Emergency Response Team / Coordination Center)

[1]UNIX is a registered trademark of AT&T

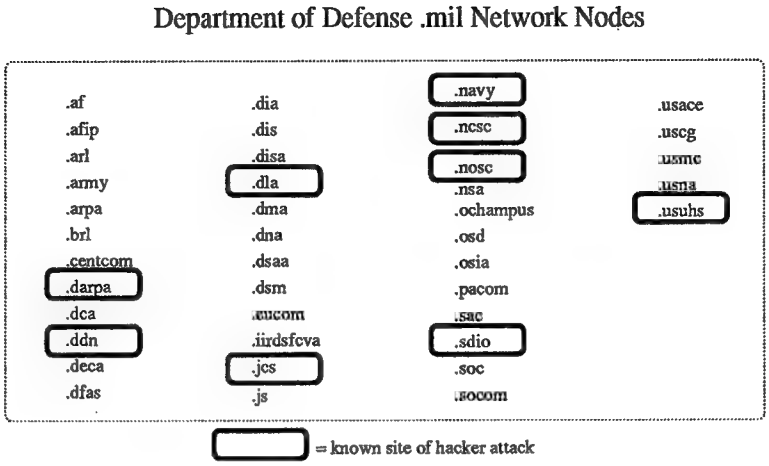

Figure 1.1 **Widespread DOD network intrusion.**

notify them" (26). Another incident report on intrusions at Columbia University states, "a major component of detection can be attributed to pure luck" (1).

Methods of detecting intrusions should be more scientific than relying on luck. The Air Force Information Warfare Center (AFIWC) has observed network intrusions on a first hand basis. Their emergency response team is responsible for dealing with Air Force wide computer attacks. Their knowledge of intrusion techniques has resulted in the formulation of attack signatures. A possible computer intrusion can be characterized as a set of UNIX commands or network activity which results in the attacker getting unauthorized access to a DOD system.

These attack signatures will form the basis of this thesis. By scanning for attacks on a regular basis, system administrators can play a more active role in securing their systems.

1.2 Problem Statement

DOD computers are being penetrated at an alarming rate. The information and systems needed to detect these intrusions are not being utilized by system administrators.

A tool is needed to help a Computer Emergency Response Team (CERT) quickly find network intruders by searching for attack signatures.

1.3 Rationale / Benefits

Computer security is of the utmost importance throughout the Department of Defense. The military has a continual reliance on information systems and networks for the processing of both classified and unclassified data. Network downtime caused by attacks costs the DOD millions of dollars and the betrayal of sensitive government information.

A 1991 report completed for the U.S. Air Force states "much research and development needs to be done before intrusion detection will be generally available" (22). It is now 1994 and intrusion detection systems are still not widely available. One of the goals of this thesis is to develop a system which is portable and easy to use. The report explains further that "by far, the most difficult task is to analyze the auditing data" (22). This thesis will continue experimentation into effectively analyzing gathered data.

1.4 Scope

The goal of this thesis effort is to create a system to detect computer intrusions in UNIX-based networks. UNIX is a common operating system for many of the workstation networks that are popular throughout the DOD. A majority of the networks connected to the Internet today are UNIX-based. This thesis is concerned with safeguarding unclassified systems which often lack the audit capabilities of classified systems.

Intrusions will be detected by filtering the user commands generated on the network. The primary method for detecting intrusions will be matching commands against attack signatures. This method will be supplemented by analyzing commands for large amounts of CPU usage, monitoring password changes, and profiling each user by his command use. Users who have commands that match the attack signatures will be reported to the system administrator. He can then take whatever action is appropriate, such as monitoring the user further or turning off the compromised account.

1.5 Methodology

Many methods of gaining access to a network or penetrating to the root level are known. The purpose of this thesis effort is to research and build a tool that actively looks for network intruders. The tool should be easy to run and provide all the necessary data a system administrator requires to assess the current security state of his network.

The program uses data collected from accounting logs of UNIX commands issued by each user on the network. The CPU time used to process each command is also part of this data. This data is analyzed in four different ways to look for intrusions.

The main intrusion detection method is to look for known attack techniques such as trying to gain root permissions, stealing password information, or denying computer service to others. The program scans all recorded commands looking for matching attack signatures.

The second method is to look for users changing their passwords. This does not appear to be a problem on the surface, but a large number of password changes could indicate a hacker present on the network. This is referred to as a *command and control problem* by the AFIWC, and will be explained in greater detail in Chapter III.

The third method reports the commands or programs that consume the most processing time. The hackers of today are using sophisticated, automatic tools such as password crackers. Many of these tools use large amounts of CPU time and can be singled out.

The fourth method used for intrusion detection is a simple account profiling system. Each day, users generate a unique profile by their use of UNIX, Internet, programming, and editor commands. The profiling feature helps the system administrator monitor users for indications of the propagation of a virus or a stolen account.

Each day the network security program runs automatically and reports its findings to the system administrator. The system administrator can then take immediate action with the results. The first method used by the program searches for intruders utilizing known attack techniques. The last three methods can be used by the system administrator to spot trends which may indicate a security breach. The program can also be used by a CERT to identify the user who is performing network intrusions.

The security program shows resistance to false positives (attacks that do not exist). By analyzing the command usage of 109 ordinary network users, the researcher has been able to show that attack signatures are limited to those people who are actually performing attack-type behavior. After one month of testing, most of the attack alarms that occurred could be attributed to the system administrator performing routine operations.

1.6 Executive Overview

Computer security must evolve to keep up with society's growing reliance on interconnected systems. System administrators are lacking the necessary tools to actively search for intruders. CERTs need additional tools to find network attackers. Most intruders are still being discovered by accident.

This thesis work provides a tested tool that searches for UNIX-based network attacks using four different methods. The program runs automatically each day and provides the system security officer with an easy-to-read security report. It will allow him or her to more easily and actively spot network intruders.

The remainder of this report explains the security tool in detail. Chapter II is a literature review discussing UNIX command logging features and documenting current research and commercial efforts in intrusion detection. Chapter III provides a top-level system design, explaining key implementation decisions. Chapter IV gives a detailed explanation of the design implementation. Chapter V reviews the collected data and presents an in-depth discussion of my findings. Finally the thesis concludes with Chapter VI, which shows the final result that a simple tool can be built to improve intrusion detection in UNIX-based networks.

II. Background

This chapter is a literature review. First, there is a review of UNIX accounting and auditing tools. Next, some advanced intrusion detection systems are presented. Each system will be presented with its features and shortcomings. With the shortcomings identified, a security tool can be build to address them.

2.1 UNIX Logging Facilities

There are two types of UNIX logging: accounting and auditing. Accounting has always been a part of UNIX. Its primary purpose was to record computer use for administrative purposes, but it can be used to aid with security. Auditing features are a recent addition to most UNIX-based operating systems. Auditing is designed to aid with enforcing computer security, but there is no standard recording format and it is expensive to implement with regard to disk space. The next two sections will describe accounting and auditing in more detail.

Filename	Meaning
/usr/adm/lastlog	Logs the last time a user logged in
/etc/utmp	Logs current users on the network
/usr/adm/wtmp	Logs the time a user starts and ends a session
/usr/adm/acct	Logs each command run by every user on the system

Table 2.1 **Four UNIX log files.**

2.1.1 UNIX Accounting. Basic UNIX log files can be used by a system administrator to identify possible intruders. The four primary UNIX accounting files which can be used for aiding with security are identified in Table 2.1 (12). The first two log files in the list store user times for accessing the system. The information provided by both these logs is duplicated in the third log. Because this duplication exists, only the last two logs in the table will be explained in detail.

The *wtmp* log can be used to identify a user logging in at a peculiar time. For example if Airman Smith works a day shift at a data processing facility, his system access times should be between 8 am and 5 pm. A login at 2 am on the weekend *could* be interpreted as an intruder masquerading as Airman Smith. This interpretation would have to be made by the system administrator reading the file or utilizing a separate program to process this information.

The *acct* log can be used to monitor commands run by the system users. The *acct* log can generate specific files by accessing it through the use of the *lastcomm* command. Accurately monitoring commands is an important part of this thesis. The basis of attack signatures is that intruders utilize commands in a certain order to exploit UNIX security loopholes. A problem with the *acct* log is that it does not record the command line switches or arguments which accompany some commands (12). Argument information is necessary to recognize some attack signatures.

The biggest disadvantage to both logs is the lack of an intelligent security monitoring capability. Separate tools are required to intelligently process these files. Another problem is the rate at which these files grow. The *acct* log alone grows at a rate of approximately 4K for each user during one day for each computer being used, which makes it impossible for an unaided system administrator to read. Many system administrators do not use accounting at all because they do not want to sacrifice the disk space which could be used for "better things".

Another form of command logging occurs at the user level. Each shell usually has an automatic command history which can be accessed through the *history* command. This shows all information that was entered at the command line including file objects and arguments.

An associated file is automatically kept for each user called *.history*. This file is updated on a sporadic basis showing portions of past command sequences. Its reliability at accurately recalling all past commands is questionable. This is because these files may be freely accessed, customized, and modified by individual users.

Another problem with using any type of files related to *history* command output for computer security purposes is that the commands must be antialiased. An alias is a UNIX feature in which a user can redefine a command to be another name. Attack signatures rely on recognizing certain command sequences, so aliased commands present a major problem.

Unlike *.history* logs, *acct* does not show aliased commands. The actual antialiased command is recorded in the log. This is a very important feature of accounting at the system level.

Command	Message	Condition
reboot halt shutdown	reboot, halt, or shutdown by <user> on <tty>	<user> used the etc/reboot, halt, shutdown to reboot, halt, shutdown the system
login	ROOT LOGIN REFUSED ON <tty> [FROM <hostname>]	root tried to log onto a terminal that is not secure
login	REPEATED LOGIN FAILURES ON <tty> [FROM <hostname>] <user>	Somebody tried to log in as <user> and supplied a bad password more than 5 times
login	ROOT LOGIN <tty> [FROM <hostname>]	root logged in
login	dialup <tty> <user>	The user <user> logged in on the dialup line <tty>
su	BAD SU <user> on <tty>	Somebody tried to su to the superuser and did not supply the correct password
su	<user> on <tty>	<user> used the su command to become superuser

Figure 2.1 **Berkeley UNIX security messages.**

Berkeley UNIX provides another logging program called *syslog*. Certain system programs can generate messages for the log. The *auth* facility in syslog monitors programs

asking for authorization. These are the commands that are frequent targets of intruders. The conditions that trigger command warnings are listed in Figure 2.1 (12).

The UNIX command *sa* provides an important security feature. It produces a list of several facts about each command including: memory used, number of times invoked, and CPU time used (8). This listing can be used to spot CPU time-intensive programs such as password crackers or be scanned for commands appearing in attack signatures.

2.1.2 UNIX Auditing. Most of the files and features described in the last section were incorporated into UNIX for accounting purposes, not security purposes. Security auditing refers to a more detailed level of monitoring the actions of system users. UNIX provides the basis for many companies' operating systems, but each company has its own version. Because of this, there is no standard auditing format or method used in UNIX systems.

However, some of these auditing methods are worth reviewing. Sun Microsystems distributes C2 Security with versions 4.0 and higher of SunOS. C2 Security is an improvement over system accounting such as that provided by the *acct* log. Besides recording the commands executed, it can record the files accessed and the success or failure of a command (10).

Digital Equipment also adds a higher level of auditing in its ULTRIX operating system. Its features are similar to the Sun auditing. In addition to Sun and Digital, there are many third-party vendors who provide their own versions of auditing packages, such as Computer Associates, OpenVision, and Systems Center (15).

The disadvantage of auditing is that the amount of information that needs to be processed exceeds that of system accounting. Each command needs a different amount of disk space to record its effects. For example, "a *find* command searching for executable files would generate megabytes of data if started in the */usr* directory" (10).

UNIX auditing is less standardized than UNIX accounting. Each operating system manufacturer has its own method. In addition, many network administrators do not use system auditing because of the extra disk space required to monitor users.

2.1.3 UNIX Logging Summary. Basic UNIX accounting and auditing features can record system activity but cannot actively find computer intruders. The amount of basic data which needs to be processed by the system administrator can easily overwhelm him. Frequently the only way to determine an attack has occurred is to watch for its effects: slow system response time, new unauthorized accounts, or file changes. Better methods are needed to monitor for intruders and automatically process the data collection logs.

2.2 Intrusion Detection Systems

The next subsections describe current research and commercial efforts in the area of intrusion detection. All the systems run on some form of UNIX operating system or may be soon be ported to a UNIX operating system. A review of current systems is beneficial for showing the capabilities that can be achieved as well as areas which require further exploration in this thesis.

2.2.1 Stalker. Stalker[1] is a misuse detection tool developed by Haystack Labs. It has three parts: a misuse detector, tracer/browser, and audit control unit (28). The entire Stalker system is run from one computer which receives its input from audit trails produced by the other computers on the network.

The misuse detector filters security data for signs of system misuse. This misuse can be attributed to either external system penetrators or legitimate users performing unauthorized actions. Stalker works by matching audit data against a large encrypted database of misuse techniques. If a match is found, the system administrator can be alerted via email or by using other notification techniques.

The tracer/browser part of Stalker allows a system administrator to selectively filter the audit data. Daily reports and queries can be customized. These queries can be complex such as, "Who was on machine X between 1am and 3am?"

[1]Stalker is a registered trademark of Haystack Labs, Inc.

The audit control part of Stalker initializes auditing for each machine on the network. The data is standardized into a form which is used by the misuse detector and tracer/browser. Audit data includes the following information: time, event, process-ID, success of event, user-ID, group-ID, session-ID, security-level, object description, and miscellaneous data (28).

Stalker is a new tool. It was first shipped in late 1993. It is too early to determine the success rate consumers have with the tool. Stalker is limited by restricting its filtering to known attack techniques. The system is expensive, with prices starting at $15,000 a network.

2.2.2 POLYCENTER Security Intrusion Detector. The POLYCENTER Security Intrusion Detector is a product of the Digital Equipment Corporation. It is a real-time monitoring program that must be installed on each machine needing protection. Audit information gathered on the host computer is filtered through a knowledge base to determine if a security violation has occurred.

POLYCENTER monitors thirteen different events. This includes monitoring login attempts, access to critical files, programs run with superuser status, and network file transfers. The goal of this filtering is to review the audit information in ways similar to an experienced computer security professional.

POLYCENTER works with ULTRIX and SunOS operating systems at a cost per node of $400. Its reliability at catching intruders is unknown. A shortcoming of this system includes no mechanism for detecting known intrusion methods.

2.2.3 NIDES. NIDES evolved from the IDES (Intrusion Detection Expert System) project at SRI. It is the result of over eight years of research in computer security. NIDES monitors a network in real-time from a host system using two different techniques: statistical analysis and rule-based analysis.

Each user of a NIDES-monitored network has a computer use profile associated with him. Every time the user accesses the network a new session profile is created. The type of behavior monitored includes: login and logout times, command execution,

directory changes, terminal use, file access, and network activity (21). This session profile is compared against the historical profile of the user through statistical analysis. If the NIDES system determines the user has changed drastically from his historical profile, an alarm is raised. This could indicate an intruder masquerading as a legitimate user (20).

The historical profile is regularly updated to provide a better picture of a constantly changing user. NIDES uses an aging process to slowly eliminate older data from the profile. Recently collected user data has more precedence in determining if the current activity is anomalous.

While the statistical analysis of user profiles can help guard a system against unknown threats, rule-based analysis of network activity is designed to uncover known intrusion methods. NIDES uses rules to identify activities known to be good indicators of attacks such as access to the password file. If a user session triggers one of these rules in the expert system, an alarm is raised.

One of the advantages of NIDES is the ability of an individual system administrator to customize the system. He can alter the parameters of the statistical analysis to prevent false positives. It is also possible to change the rule base of the expert system to identify a new type of attack. These features are lacking in other intrusion detection systems (20).

NIDES is currently in the beta test phase. The only places this system has be used is at SRI International where it was developed and at the FBI (13). Justin Lister, a researcher at the Center for Computer Security Research at the Univerity of Wollongong commented that, "Considering the amount of research in the IDES/NIDES systems, I think we can safety assume that it will never be available to the masses (I may be wrong), but I would think the complexities of the system would ensure it is out of reach of the less experienced system administrators" (18). Lt Roth from the AFIWC also has doubts about the ability to implement NIDES on most networks. "Everyone runs to avoid this tool. It is based on anomaly detection, which currently does not work. It has a statistical based approach as opposed to a signature based approach – which does not appear to be as good" (24).

2.2.4 Multics Intrusion Detection and Alerting System. The Multics Intrusion Detection and Alerting System (MIDAS) is a real-time intrusion detection system used to watch over the Dockmaster network at the National Computer Security Center. It is implemented on a Honeywell Multics mainframe to monitor the system's 1,200 users (27). The security system uses both statistical inference and expert systems technologies.

MIDAS uses a number of sources to detect intruders. User commands and network statistics are preprocessed into facts which are fed into the expert system engine. The expert system uses current facts to trigger rules stored in a rule base. If an suspected intrusion is detected, MIDAS will print a message on the system administrators console.

Rules are broken into three areas: immediate attack, user anomaly, and system state (27). Immediate attack rules concern themselves with direct attack sequences. An example of this is a normal user attempting to use superuser commands. User anomaly rules analyze user sessions. Every time a user logs in, a session record keeps track of his or her actions. These actions are used to modify a user profile record. If a particular user session varies drastically from the usual profile, this will trigger a user anomaly rule. System state rules look at the overall network. These rules can notice large trends such as a dramatic increase in processing time across the entire network. This could indicate that an intruder is using large amounts of CPU time to run a password cracking program.

MIDAS is powerful. By utilizing these three distinct rule areas, it can detect break-ins, masqueraders, penetrations, misuse, and Trojan horses (27). One disadvantage of MIDAS is that it has only been used at the National Computer Security Center. Another disadvantage is that it was written with an in-house expert system. This limits the ability of MIDAS to be portable and modifiable by others.

2.2.5 Distributed Intrusion Detection System. DIDS (Distributive Intrusion Detection System) is the joint effort of the University of California at Davis Computer Security Laboratory, Haystack Laboratories, and the U.S. Air Force Cryptological Support Center. It is designed to monitor a heterogeneous network of computers. DIDS uses a unique approach to intrusion detection by monitoring at both the local and network levels.

Each computer on the network contains a host monitor. The host monitor filters audit data from only the computer on which it is located. This filtering includes looking for attack signatures (29). The purpose of the attack signature detector is to try to identify certain command usage as attack-type behavior. An example of this is a user attempting to erase all the system files.

Some audit data which is helpful for analyzing local events is also helpful for tracking system-wide behavior. If the host monitor finds occurrences of notable events, such the use of network access commands or requests for user authentication, these events are forwarded to the DIDS director and not handled on the local level (30). Periodic summaries of security conditions at each host are also sent to the DIDS director.

There is also a local area network (LAN) monitor for the network. This monitor examines Ethernet packet streams for suspicious port accesses. It can also be used to help monitor computers on the network that do not contain host monitors. Information about the security state of the network is forwarded to the DIDS director.

The DIDS director is located at one computer on the network. It receives security information from all the machines on the network via communication with each individual host monitor. It also receives input from the LAN monitor.

The DIDS director utilizes a rule-based expert system to reason with the collected information. Through looking at the network as a complete system, DIDS can find network-wide attacks that would go unnoticed at the host level. Events that occur on the network are combined with a context to produce a threat (30). For example, a password change may not be a notable event, but when everyone changes their password at 3am one morning, the events are considered threats.

The top level output of the DIDS director is a number from 1 to 100 which indicates the current security level of the system. A higher number is used to show high levels of attack or misuse behavior on the system. The system administrator can monitor this number and query the DIDS director for more information about the notable security events.

DIDS has a similar history to the NIDES system. There is only one other user of DIDS besides the development team. Since this is a DOD funded project, this thesis effort is designed to provide a better understanding of some of the concepts that are incorporated into DIDS.

There are three aspects of the DIDS implementation that I wish to explore. First, DIDS has a requirement of utilizing extensive auditing information. Earlier in this chapter it was shown that many system administrators do not have extensive auditing capabilities available, only general accounting information.

This thesis will attempt to use only those features provided by basic UNIX accounting. It can be argued that more information is better for good security, but this thesis will show that most users do not use or even stumble across attack-type commands. These commands will not appear so frequently that it is impossible to determine an attack from a non-attack. Security may be kept by monitoring only the commands themselves for matches with attack signatures.

The second aspect of the DIDS system which this thesis will investigate is how the users' sessions are treated. Current context is lost in DIDS if a user exits the system. Therefore a user could set up his attack and log off. Then he could log back on and complete his attack without detection. The system built for this thesis will preserve the context of a user across sessions as well as across multiple machines, provided that the user logs in under the same account name.

The last aspect of the DIDS system relates to the attack signatures. Both the system implemented in this thesis and the DIDS system are based on a similar set of attack signatures (25). By independently developing an intrusion detection system based on the same attack signatures, further insight and understanding into the attack signatures themselves is achieved.

2.3 Current Security State

The review of intrusion detection systems indicates this area is still in its infancy period. Intrusion detection systems are not being widely used. A representative from the

Defense Information Systems Agency's emergency response team stated that most system administrators use no monitoring or system accounting of any kind, some use UNIX basic accounting, and only a handful of sites use any type of dedicated intrusion detection system (19).

The current state of security for many unclassified UNIX-based networks is poor. Millions of passwords for these systems have already been compromised through the use of network sniffers (3). If passwords have already been comprised, then there is no alternative but to use other forms of security to protect the network.

2.4 Summary

Current intrusion detection systems are still very experimental and not widely used. Most systems were installed at only one research and development site. These tools appear competent for detecting intruders, but proven security at multiple sites is lacking. Also, the cost of many of the intrusion detection systems may be prohibitive for implementation on some networks.

While these systems continue to be developed, unclassified DOD networks around the globe continue to be easy targets for intruders. The effort of this thesis is to research and develop a system based on standard UNIX accounting features that can aid system administrators and CERTs in finding intruders. It is intended to bridge the gap between no security and an advanced security system.

III. System Design

This chapter will explain the upper-level design of the intrusion detection system built for this thesis. A set of goals were determined to define the intended system. With clear goals, implementation decisions were made to provide the framework on which the system is built.

3.1 System Goals

The goals in developing the security system are to make it easy to use, portable across unclassified UNIX-based systems, and effective in detecting network intruders. For ease of use, the system administrator should not need to interact with the system constantly to insure correct behavior. He should be able to check his terminal or read a file and understand the current state of the network.

The next goal for the intrusion detection system is to be portable across UNIX systems. The implementation of the design needs to consider the resources available to UNIX system administrators. This portability requirement applies to picking an implementation language and an accounting or auditing system. Portability is especially important to a CERT that travels to different computer sites around the globe.

The third goal is for the system to be effective at detecting intruders. There are many different implementation methods to choose from to accomplish this task. One can monitor at the packet level and watch every packet going over the network. Another method is to monitor keystrokes and watch every single keypress by network users. This thesis will be restricted to techniques that involve analyzing data at the command level. The attack signatures found in Appendix A are the main focus of this research, so this is the level at which the security system will process information.

3.2 Implementation Decisions

Four implementation decisions form the basis of the security system design. Each decision was made after considering all of the intended goals of the system. Design choices

that needed to be resolved include: obtaining command information, processing command information, detecting intrusions, and presenting the security data.

3.2.1 Obtaining Command Information. The first implementation decision was deciding where and how to obtain command information. Chapter II discussed some different methods of gathering this information. The *history* files associated with each user provide a wealth of information including command name, line switches, and file access, but the shortcomings far outweigh the benefits of using this method. Each window open on a user's terminal has its own *history* file associated with it. The time a command is executed is not recorded so comparing and combining commands from different windows from a single user would be impossible. This is a necessary feature for detecting time-dependent attack signatures.

Another method that was explored is the use of a secure shell to automatically record the commands entered by a user. The commands could be time ordered as they are sent to a separate file to be recorded. This method has still suffers from some of the problems mentioned in Chapter II such as the need for additional antialiasing features. It also requires all system users to use the secure shell.

The method which was finally chosen for obtaining command information is the use of the */usr/adm/acct* log which is processed by the *lastcomm* command. Although line switches and files access is not recorded, it does provide a time-ordered list of user-issued commands. Current UNIX security literature also recommends this log as a good place to look to monitor command usage (7, 10, 11, 12, 15). Auditing was not used because there is no standard format available across the different UNIX implementations. One of the goals for the system is portability, and utilizing system accounting information meets this goal.

3.2.2 Processing Command Information. The second implementation decision was deciding what technique to use to process the data presented in the accounting log. First an expert system based in CLIPS (C Language Integrated Production Systems) was used to filter the commands against the attack signatures. CLIPS is a programming

language developed at NASA for developing rule-based systems (14). Powerful pattern-matching programs can be quickly developed with CLIPS.

This method was determined to be overkill because without command switches or file names, only a simpler form of pattern-matching was required. Instead of CLIPS, C code and UNIX shell scripts form the basis of this security system. This aids in making the system both portable and simple to modify.

3.2.3 Detecting Intrusions. The third implementation decision required focusing on specific areas of intrusion detection. All the information in the *acct* log had a potential for being used. This includes the command, username, CPU time used, and time the command was issued. After reviewing other intrusion detection systems, it appeared wise to include features that could be used to detect both known and unknown attack situations. Known attacks methods are covered by scanning for attack signatures in a method similar to DIDS. Attack signatures provided by the AFIWC are listed in Appendix A and form the basis of this security program.

Three techniques are used to find intruders who do not match the attack signatures. The first technique is to bring password changes to the attention of the system administrator. A password change in itself is not an attack behavior, but could be the signal of an intruder taking over an account. This is further detailed in Chapter IV.

Another technique is to monitor commands which utilize enormous amounts of CPU time. Password cracking programs are favorites for hackers wishing to gain system access. These programs require a great deal of computing power. The current method of watching for cracking programs involves a system administrator running the *ps* command on each machine he wishes to monitor. This command will list the current processes running on the computer. It is an impossible task for the system administrator to run this command on all the network computers throughout the day. The security program developed for this thesis will provide a much easier way for making this task automatic and painless.

The third technique used to help find intruders is the use of user profiles. The security system will use a simple method of profiling the commands utilized by each user. The profiles provide the system administrator with quick breakdown of a user's activity.

A sample profile can show that User1 ran network, mail, and programming commands during Day 5. If User1 is not a programmer, the account may have been compromised by an intruder masquerading as User1.

3.2.4 Presenting the Security Data. The final implementation decision was deciding how often to process the data and how to present it to the system administrator. The security system runs in batch mode every twenty-four hours. All the accounting data gathered during the day is processed at this time and a comprehensive report is mailed to the system administrator.

The shell scripts can be modified to run every two minutes. This will not increase the number of missed attacks because the *lastcomm* command has a built-in memory capability. All commands executed after a fixed point in time will be recorded. With a 24-hour implementation, this means a fresh log is started every day, and it will grow every two minutes. By processing the log file every two minutes, execution time will get slower as the day progresses because the file size will increase. If speed is crucial, then the security program will need to be modified to save only those commands matching parts of attack signatures and not every command executed. As the program is setup now, *lastcomm* has the necessary memory capability to record commands which executed during any previous two-minute time span.

By processing data every 24 hours, the network security officer does not have to face a deluge of data every other minute. If the user needs the system to run in real-time, the security message would need to be tailored to only report recent attack signature matches. Even with real-time operation, it is important to remember that a network security officer may not be able to react in enough time to stop the attack. He may be away from his terminal for an extended period of time.

This daily reporting method is used by AT&T in a similar network monitoring application (6). Stalker, NIDES, and DIDS all provide the ability to produce daily reports delivered by email to the system administrator. This lessens the workload of the system administrator by presenting him with a detailed report of all security-related events from the previous day.

3.3 Summary

All implementation decisions were made to satisfy the system goals. The system is designed to be easy-to-use by presenting all computer security information in a daily electronic message. The system is portable because it is coded in C and UNIX shell scripts and the accounting data is obtainable on any UNIX network. Finally, the system is designed to detect intruders by utilizing four different intrusion detection methods. This upper-level design is explained in more detail in the next chapter.

IV. Design Implementation

This chapter details the implementation of the top-level design. First, the data collection method is presented. Next, the data filtering process is reviewed. Third, the data is analyzed to determine its suitability for the task of scanning for attack signatures. When this step is completed, intrusion detection features are added. Next, a user profile method is created to supplement the intrusion detection. Finally, the intrusion detection methods are combined with the user profile to form a working security system.

4.1 Data Collection

The system uses accounting log data from two computers on a local area network. A full implementation of this system would require data from all the nodes on the network. Since this is a research system, only two nodes were monitored. The audit data is generated by using the *lastcomm* command as an interface to the *acct* log which was discussed in Chapter II. This command produces a list of all the commands issued to the computer. This listing has eight columns: command name (first eight characters), flags (indicating whether the command process forked, set-user-id, dumped memory, or was killed), username, terminal type, CPU process time used in seconds, day, date, and time when the process started. A portion of the accounting log is presented in Table 4.1.

There are many benefits to using this type of accounting information. Multiple users can be tracked simultaneously. The same user can also be tracked across multiple machines. Aliased command names are resolved and shown in their true form. Shell scripts are broken down to the command level and recorded.

4.2 Data Filtering

Data filtering is accomplished through a series of shell scripts utilizing the UNIX *cut, paste,* and *sort* commands. Two different ways to preprocess the data were explored. First the data was collected on a weekly basis. The accounting information from both computers was merged into a single file so the information could be analyzed as a mini-network.

Command	Flags	Username	Terminal	Cpu Time	Day	Date	Time
csh	S	dlandry	__	0.66 secs	Mon	Sep 26	16:40
in.rlogi	S	root	__	0.75 secs	Mon	Sep 26	16:40
clear		dlandry	ttyp1	0.02 secs	Mon	Sep 26	16:47
rn		dlandry	ttyp1	0.64 secs	Mon	Sep 26	16:46
rn		dlandry	ttyp1	0.61 secs	Mon	Sep 26	16:46
rpc.rsta		root	__	0.02 secs	Mon	Sep 26	16:45
in.tname		root	__	0.05 secs	Mon	Sep 26	16:38
rlogin		dlandry	ttyp1	0.03 secs	Mon	Sep 26	16:42
csh	F	dlandry	ttyp1	0.00 secs	Mon	Sep 26	16:42
rlogin		dlandry	ttyp1	0.03 secs	Mon	Sep 26	16:41
passwd		dlandry	ttyp1	0.03 secs	Mon	Sep 26	16:41

Table 4.1 **Unfiltered lastcomm generated accounting log.**

This file was reduced by uniquely sorting by username. This is necessary to speed the processing time of the file by the intrusion detection program. Unique sorting eliminates duplicate lines. This shortens the file drastically while preserving the content.

A good example of this benefit can be drawn from observing the data collected in the logs. In one case, someone had written a shell script to check the printer status every three seconds. This resulted in *lpq* showing up about 50 times in the accounting logs. After unique sorting, it would only show up a few times based on processor time used.

The disadvantage to unique sorting is that some instances of user activity are completely eliminated before being run through the intrusion detection system. If intrusions are occurring, a system administrator would want to know about all of them.

The current method of filtering involves collecting the data on a daily basis and sorting (not uniquely) by username and time of day. By dealing with the data on a daily basis, the length of the file is kept at a manageable level. Sorting by username and time groups the data from the two machines in a precise manner which is easier for a human to comprehend. If the output from the intrusion detector is ever in question, the system administrator can go back to the logs and trace one user's actions step by step, regardless of how many machines he utilized.

In Table 4.2, the data from Table 4.1 is now filtered. It is sorted by username and time of day.

Username	Command	Cpu Time	Day	Date	Time
dlandry	csh	0.66	Mon	Sep 26	16:40
dlandry	passwd	0.03	Mon	Sep 26	16:41
dlandry	rlogin	0.03	Mon	Sep 26	16:41
dlandry	csh	0.00	Mon	Sep 26	16:42
dlandry	rlogin	0.03	Mon	Sep 26	16:42
dlandry	rn	0.61	Mon	Sep 26	16:46
dlandry	rn	0.64	Mon	Sep 26	16:46
dlandry	clear	0.02	Mon	Sep 26	16:47
root	in.tname	0.05	Mon	Sep 26	16:38
root	in.rlogi	0.75	Mon	Sep 26	16:40
root	rpc.rsta	0.02	Mon	Sep 26	16:45

Table 4.2 **Filtered lastcomm generated accounting log.**

4.3 Feature Selection

Filtering removed the flags and terminal columns from the accounting logs. Also the host machine names of the two computers that comprise the mini-network is omitted. In this security system, this additional information was not needed. Eliminating unnecessary data decreases the processing time and reduces the amount of space needed to store accounting information.

These eliminated items would have provided marginal security improvement at best. The flags provided no more revealing information about a user. The flag which indicates a user ran a set-user-id program may seem useful for security, but it occurred too often in normal everyday computer use patterns.

This security system was implemented in an open-lab environment where users show no real attachment to any particular computer or terminal. The machine name and terminal type could be important in another type of environment such as an office-to-office network where each user uses his own private workstation.

The features that were kept include name, command, CPU time, date, and time. CPU time was determined to be important after reviewing some security incidents such as those at Columbia University. Password crackers were utilized in three of the five security breaches at Columbia (1). All three of these attacks were discovered because the password cracking programs were using large amounts of CPU time (1).

4.4 Data Analysis

A study of UNIX command use is necessary before detecting attack signatures or organizing a user profile. This analysis insures a firm understanding of the type of data being collected. The sample data for the study was gathered over a period of ten days. Fourteen commands emerged as being used by at least half of the 39 users. The results are listed in Table 4.3.

Percentage of 39 users	Command	Definition
86	biff	Give Notice of incoming Mail Messages
89	cat	Concatenate and Display
89	clear	Clear the terminal screen
98	csh	c-shell
83	date	Display or set the date
84	echo	Echo arguments to the standard output
88	hostname	Set or print name of current host system
68	ls	List the contents of the directory
87	quota	Display a user's disk storage and usage
85	sed	Stream Editor
96	sh	Standard UNIX shell
70	stty	Set or alter the options for a terminal
86	tty	Display the name of the terminal
83	whoami	Display the effective current username

Table 4.3 **Most common commands in accounting log.**

These results are somewhat surprising, because an informal study conducted by the researcher of twenty random users' *history* files indicated that other commands would be more frequently used such as *rm* (remove a file), *cd* (change directory), and *logout*

(terminate a login shell). The discrepancy arises from the fact that some commands are issued automatically as the user interacts with the system. So in effect, some commands are shielded from the user's knowledge. Examples of these automatic commands include *biff, clear, quota,* and *hostname.* The results of the data analysis is encouraging. The most frequent commands do not match attack signature commands, which indicates that the security program should be able to find attacks without being sidetracked by unnecessary noise.

4.5 Intrusion Detection

The intrusion detection part of the security system is a pattern-matching program written in C. The accounting log is scanned for attack signatures, programs consuming large amounts of CPU time, and password changes. Attack signatures are matched against one user at a time.

The time ordering is important in the recognition of attack signatures. The intrusion system uses the *follows* semantic highlighted by Kumar and Spafford (16). "For example, with the "follows" semantics the pattern *ab* specifies the occurrence of the event *a followed* by the occurrence of event *b* and not (necessarily) *a* immediately followed by *b* with no intervening event" (16). If attack-type commands occur in a non-threatening order then no attack will be recorded. Scanning for attack signatures is the main method used in this thesis for detecting intruders. The reader should review Appendix A now to fully understand how this method was implemented.

The time and command name of the process consuming the most CPU time is recorded for each user. This feature helps the system administrator in watching the daily load across the system. It can also be used to quickly find system abuses such as a password cracker running under different usernames (1, 10).

Another feature useful in detecting intruders is monitoring password changes (21). When an intruder takes over an account, he frequently changes the password (1, 21). Password changes are also important with regard to information warfare. A large group of

users changing their passwords could indicate a hostile takeover of the computer system. Control of a valuable unclassified system could easily and quickly be lost.

4.6 User Profile

An additional step to aid intrusion detection is the use of a user profile. The profile attempts to discretely measure each user's UNIX and network proficiency through the use of certain commands. When a system administrator reviews the security report he can quickly understand exactly what kinds of tasks a particular user was performing during the day.

The user profile has nine categories: Common, Unix1, Unix2, Mail, News, Information, Editors, Programming, and Internet. Each category is representative of certain commands and programs found in the UNIX environment. For a given day, a user belongs to a category or does not. For example, if no programming languages are used, the user's profile will not contain the Programming category. This binary method of reducing large amounts of audit information is used in a similar manner for an intrusion detection system developed at Tulane (17).

The categories were initially determined by reviewing current security literature and UNIX books. The categories and commands are listed in Table 4.4. The Common category is made up of all the commands listed in Table 4.3. Most user sessions use at least one command in this category.

The commands chosen for each category took a couple of weeks to determine. Some commands were switched to different categories based on the amount of use observed during the data analysis phase. For example, *uname* is an information command but it appeared so often in the accounting logs that it was moved to the Unix1 category. The *grep* command was also moved from Unix2 to Unix1 because of the frequency of use.

The Programming commands were also altered when it was determined that some commands do not necessarily map directly into the accounting log. For example, the command to compile a program, *c++*, is broken down into a few more basic commands which do not include *c++* in any form.

Group Name	Command Type	Commands
Common	Most frequently occurring	biff, cat, clear, csh, date echo, hostname, ls, quota, sed, sh, stty, tty, whoami
Unix1	Basic UNIX	cp, lp, lpq, lpr, mkdir, mv, pwd, rm, rmdir, uname, grep, more
Unix2	Advanced UNIX	chmod, find, awk, ps, mps, kill, sort
Mail	Electronic mail handling	elm, mail, mailtool, sendmail
Programming	Programming languages	cpp, cc1, gcc, as, lisp-full, lisp-fla
Information	Information providers	finger, rusers, users, w, who, whois
Internet	Network access	ftp, telnet, rlogin
Editors	Vi and Emacs editors	vi, view, vedit, ex, e, edit, emacs
News	Newsgroup readers	nn, rn

Table 4.4 **Commands contained in user profile categories.**

The Unix1 category is composed of the basic UNIX commands. These commands can be found in beginner UNIX books. The Unix2 category is composed of higher-level UNIX commands. These are commands associated with a more skilled UNIX user and can be found in advanced UNIX books.

The purpose of segmenting the commands into categories is to provide a quick visual check for the system administrator. If Airman Jones is an administrative clerk, his profile would probably include the categories Common, Unix1, Mail, and News. If the Programming category shows up, a security breach of Airman Jones' account may have occurred.

The classification of commands is designed to provide individual user profiles that can be contrasted against other user profiles or the same user for a different time period. The classification is not intended to cover all possible UNIX commands or programs.

4.7 Security Report

The intrusion detection techniques are supplemented with user profiling to present a daily security report to the system security officer. A sample report is presented in Table 4.5.

```
--- Matches trusted hosts Attack Signature ---    dlandry        su
*** Password Changed for **********************    dlandry     passwd
```

username	Com	Unix1	Unix2	Mail	News	Info	Ed	Pro	Int	HI-command	HI-CPU
dlandry	X	X	X	X	X	X		X	X	pcrack	243.95
user1	X		X							movbnk.r	376.80
user2	X	X		X			X			emacs	4.75
user3	X	X		X						mail	9.30
user4	X	X	X							matlab	2.12
user5	X	X	X			X			X	ftp	1.56

Table 4.5 **A portion of a security report.**

By reviewing the report, the system administrator or CERT can understand the activities of each user on the system. In the example above, user *dlandry* has triggered an attack signature by using the *su* command and changed his password. Two users have run commands consuming large amounts of CPU time. The system administrator may wish to investigate *dlandry* first, because he has triggered an attack signature as well as run a time-intensive program, possibly a password cracker. The security tool developed in this chapter takes thousands of lines of command information and presents a comprehensive summary of them. Chapter V will review the testing of the system with actual accounting information and planned, controlled attacks.

V. Results

This chapter covers the results produced by running the security system. The two-computer mini-network was monitored for a period of one month. During this time, the system collected approximately nine megabytes of accounting data. This includes 145,066 completed commands. During this time span, 109 different users utilized the mini-network.

This chapter has three primary sections. The controlled testing section provides information about attacks that the researcher set up. The uncontrolled testing section provides information about attacks that were not staged. The accounting log shortcomings section describes some difficulties that were discovered during testing.

5.1 Controlled Testing

Each attack signature described in Appendix A was attempted during controlled testing. The purpose of this phase of testing was to insure the detection program worked properly at recognizing attack signatures. Some controlled attacks were also performed by different users.

Both of the techniques used for the rdist attack signature were recognized. However, the *ln* command was occasionally missed because of a data filtering problem. This is further discussed in Section 5.3. This problem is minimal because the *ln* command is the last phase of this attack and the system administrator would already be notified that the rdist attack has started.

The loadmodule and mount attacks were attempted and detected by the security program. Trusted host attack signatures were successfully executed and detected. The xkey attack signature was detected and recorded in the accounting logs, but the actual attack was prevented by the operating system. The loadkeys attack is similar to the xkey attack. It was detected and recorded, but the actual attack was unsuccessful.

The crash attack was also attempted but access to this command was restricted on the network. This restricted access prevents the attack attempt from being recorded in the accounting log. This is an unfortunate side effect, because attack attempts should be monitored as well as successful attacks. This is further discussed in Section 5.3.

Other tests performed during controlled testing were password changes and CPU time-intensive programs. The security program successfully detected each password change which was attempted. A CPU time-intensive program which counted infinitely was used to simulate a password cracking program. It was successfully detected and reported in the daily security report.

5.2 Uncontrolled Testing

This section covers the free actions by the users of the system. None of the users on the mini-network knew they were being monitored, so testing bias towards the detection program was eliminated. If a rogue hacker was lurking on the mini-network, the detection program was expected to find him. Two attack signatures were triggered: mount and trusted hosts.

The mount signature was detected three times. All three times were attributed to the system administrator: two times under his username and once as root.

Both versions of the trusted hosts attack signature were detected. One user performed the *su* (assume-user-identity) command four times on a given day. The user was the system administrator. Eleven *rsh* (remote shell) commands were issued by three different users.

These triggered signatures are not malicious because the system administrators of the network allow users to perform trusted hosts commands. However, it should be noted that current security literature recommends against allowing trusted hosts because this is a security weakness exploited by the infamous Internet worm of 1988 (5, 9, 12). With only fifteen uses of trusted hosts commands in one month, it appears these commands were not being abused by anyone within the mini-network.

Other methods of intrusion detection were tested in addition to attack signatures. This includes password changes, CPU time-intensive commands, and user profiles. Only 3 of the 109 users changed their passwords. It appears to be a normal rate with no indication of massive hostile account takeovers.

The recording of CPU time-intensive commands was helpful in understanding daily network load of completed processes. A few users utilized commands using large amounts of

CPU time, but all could be attributed to normal computer use. The *matlab* mathematical program appeared most often. *Matlab* program runs frequently consumed between 10,000 and 100,000 seconds of CPU time.

99.06% of all the monitored commands were completed in under 10 seconds. 99.87% of all the monitored commands were completed in under 100 seconds. These findings offer statistical evidence that CPU time-intensive processes such as password crackers can be singled out in this way. This narrows an interested system administrator's search down from thousands of commands to less than 100.

For example, during the one month of monitoring, 40 people ran processes exceeding over 10,000 seconds of CPU time. If we assume that the average password cracking program consumes greater than 10,000 seconds, than 1.33 programs would need to be checked per day for every 2 computers. Using these percentages for estimation, a system administrator would only have to check 33 programs for suspicious activity per day for a 50 computer network. Of course, these numbers will vary according to network use and type of programs run.

Another feature used to improve security was account profiling. A profile of the entire system is shown in Figure 5.1. With statistics such as these gathered for the target network of the security system, a system administrator can more accurately spot trends which indicate attack-type behavior. For instance, the Internet virus of 1988 disguised itself under shell command names. If any category besides the Common command category is recorded for every user on a given day, a virus may be present on the system, worming its way through user accounts.

The profile of the entire system also shows the original categorization of the commands was well planned. 108 of the 109 usernames were members of the Common command category (only *nobody*, a generic system username, was not a member). About half the users utilized advanced UNIX commands which was less than the number who used basic UNIX commands. Only two thirds of the users were indicated as using mail. This is lower than expected, indicating that the possibility that some mail related commands were omitted from the profile.

Percentage of 109 users	Group Name	Command Type
99	Com	Most frequently occurring
83	Unix1	Basic UNIX
66	Mail	Electronic Mail handling
54	Pro	Programming languages
48	Unix2	Advanced UNIX
38	Info	Information
37	Int	Internet
17	Ed	Vi and Emacs editors
9	News	Newsgroup readers

Table 5.1 **Profile of total system activity.**

The individual user profile proved to be very helpful in understanding how each user is utilizing the system. A masquerader was discovered by use of the profile. After reviewing the security report one day, I noticed the profile of a friend showed the use of programming languages. I knew he had not been programming lately so I asked him about the profile. He confessed to loaning his account to another user.

5.3 Accounting Log Shortcomings

During testing some unforeseen problems appeared that hindered the effectiveness of the detection program. Most can be attributed to the accounting logs, not the detection program. Three problems are directly connected to the accounting log: mismatched commands and usernames, hidden commands, and inaccessible commands.

One very subtle but annoying problem is that some commands are not always attributed to the correct username. Occasionally the command is recorded in the log as a five digit user id number, not the username. It is unclear why this condition occurs. It is not confined to certain users or certain commands. During the one month of testing, this error occurred approximately once every four thousand commands.

The most noticeable shortcoming of the accounting logs is the lack of ability to truly record all commands. This problem prevents detection of four attack signatures:

expreserve, sync, brute force, and ftp. Part of the problem is that commands which are built into the system shells are not recorded individually, but as the shell name. The c-shell (*csh*) contains commands such as: *alias, cd, kill, nohup, set,* and *setenv.*

By hiding under the shell name, these commands cannot be matched against attack signatures. There are 48 commands in the *csh* command-interpreter, 24 commands in the *sh* command-interpreter, and 55 commands accessible through the *ftp* command. A similar situation exists with the *crash* command.

To illustrate the problem, an example is given in Table 5.2. First the date command is entered. Next, some common c-shell commands are performed. Finally the session is ended with another date command.

```
output produced by history (oldest event to most recent event):
---------------------------------------------------------------
   654  date
   655  alias jj setenv
   656  echo me
   657  set
   658  setenv
   659  unalias jj
   660  cd ..
   661  cd dlandry
   662  chdir logs
   663  cd ..
   664  date

output produced by lastcomm (most recent event to oldest event):
---------------------------------------------------------------
date           dlandry  ttyp2      0.02 secs Fri Oct 28 15:05
sed            dlandry  ttyp2      0.02 secs Fri Oct 28 15:05
csh       F    dlandry  ttyp2      0.00 secs Fri Oct 28 15:05
hostname       dlandry  ttyp2      0.00 secs Fri Oct 28 15:05
sed            dlandry  ttyp2      0.02 secs Fri Oct 28 15:05
csh       F    dlandry  ttyp2      0.00 secs Fri Oct 28 15:05
hostname       dlandry  ttyp2      0.02 secs Fri Oct 28 15:05
sed            dlandry  ttyp2      0.05 secs Fri Oct 28 15:05
csh       F    dlandry  ttyp2      0.00 secs Fri Oct 28 15:05
hostname       dlandry  ttyp2      0.02 secs Fri Oct 28 15:05
date           dlandry  ttyp2      0.02 secs Fri Oct 28 15:04
```

Table 5.2 **Lost information in lastcomm generated log.**

The same data is contrasted by showing the output of the *history* command followed by the output of the *lastcomm* command. By comparing the sections between the date

commands, you will see that they are not alike at all! The *lastcomm* command is not providing information about the actual command which was executed.

A third problem occurs when an intruder attempts to use attack commands that he does not have access to. The attack is unsuccessful, but the attempt will not be recorded. However, when a command is accessible to the user, but the user is prevented by his privileges from exploiting its negative features, the command is recorded. This provides a limited capability to watch for system probers. A system administrator can see who is trying some attack signatures, but may completely miss a potential intruder *banging on the locks* (7). By not recording all intrusion attempts, some intruders can go unnoticed.

Another problem occurs in the data filtering state. During this stage the accounting information is ordered by name and then by time. The time recorded is only precise down to the minute. It is possible for many commands to be executed within the span of a minute. The UNIX sort command alphabetizes all the commands with a given username and minute. Thus, the order of commands during a minute may be jumbled.

This problem has no easy solution. It is possible to use another sorting routine, but when the data is combined from two or more computers it is impossible to determine which command actually occurred first during a particular minute. An auditing program should be used that records time to the nearest second. It is also important to keep all the computers on the network synced to one master clock so events which occur on different computers can be processed in true temporal order.

5.4 Results Summary

The security system was tested through the use of controlled and uncontrolled techniques. Attacks were detected and the number of false positives was small. Only 18 commands in the 145,066 tested resulted in the matches with attack signatures. Testing also revealed some shortcomings that were caused by accounting log deficiencies.

VI. Conclusion

This thesis effort produced and tested a security system for intrusion detection. The system is designed to help protect unclassified UNIX-based networks from attacks. It is written in C and UNIX shell scripts which make it portable, understandable, and modifiable by system administrators. It can automatically identify a majority of the attack signatures provided by the AFIWC. Its other features, which include user profiles, password change notification, and a CPU time-intensive process report, aid in finding other intrusive behavior.

The results of this thesis are characterized by two main contributions to the area of computer security and intrusion detection. They are detailed in the next section. The second section of this chapter provides some thoughts about future work in this area. Finally in the last section the researcher provides some closing comments about computer security.

6.1 Contributions

The first contribution is that my system demonstrated that even with less than ideal accounting information, a practical pattern-matching intrusion detection system can be built to find intrusive behavior. Normal network activity can be effectively filtered for attacks. There was a small number of false positives recorded, with only 3 of 109 users being flagged for their command use.

The system testing also showed which of the attack signatures could be detected using accounting information, and thus provided a better understanding of the signatures themselves. System administrators and CERTs now know which attacks can be detected through standard UNIX accounting methods and which attacks can successfully go unnoticed.

My practical security tool also found an actual security violation. The user profiling system provided enough information to determine that a account had been compromised. The more I used the tool, the more familiar I became with users and their working styles.

I knew who ran time-intensive commands and which users were more likely to generate false positives for attack signatures.

The second major contribution is that I found some previously unreported problems with the accounting log generated by the *lastcomm* command. These problems hindered the ability of my detection program to find all attack signatures. This is an important contribution to the field of computer security, because many references written by knowledgeable people in the areas of computer security and the UNIX operating system are still emphasizing that the *acct* log is a suitable place to watch command use (7, 8, 10, 11, 12, 15).

The shortcomings discovered during this thesis were not mentioned in any security reference. All the references which mentioned this form of accounting stated that this source could be used to monitor command use. The contrasting example in the last chapter showed how wrong this statement can be.

One book even went so far as to say, "... you can use this command to look for suspicious commands being run, such as *sz /etc/password*, or anything having to do with the */etc/password* file" (31). File names are not even stored in this accounting file! I have already contacted two publishers to encourage them to fix these misleading statements in their books. One author responded to a message about my accounting logs shortcomings with, "That stuff is worth clearing up in UPT (his co-authored book - UNIX Power Tools)." The author of another book also said he would add the information to the next edition.

By getting this information out to security system implementors, greater awareness is enhanced. This finding will encourage others to use a more reliable forms of auditing. This is also beneficial to other computer security scholars who may be basing some of their research on output generated by the *lastcomm* command.

6.2 Future Work

A good first step would be to test and modify the system to work with a better data collecting package. This thesis shows that UNIX accounting files generated by the *lastcomm* command are not adequate for detecting all attack signatures. The Air Force Information

Warfare Center would like to see future research dealing with signature detection performed at the packet level.

Each different implementation of UNIX has a programming capability that allows programs to read all the Ethernet traffic. SunOS has the Network Information Tap (NIT). ULTRIX uses the Packetfilter and BSD systems have the Berkeley Packet Filter (BPF) (8). By monitoring all the packets as they pass along the network, shortcomings such as those mentioned in Chapter V will be prevented because all commands and file names will be available.

If packet level information is used, the researcher must perform another data analysis to determine exactly what kind of data is available from these TCP/IP taps. For instance, the packets are often in hexadecimal form. If every packet needs to be translated to ASCII to compare to the attack signatures, the intrusion detector may not be able to keep pace with the amount of packets on the network. It may be faster for the researcher to encode all the signatures into hexadecimal to do the comparison.

Also, any future systems derived from this thesis should be able to handle all of the Ethernet forms (NIT,BPF,etc.). This will keep the system portable across UNIX-based networks for use by a Computer Emergency Response Team. In addition, a CERT will need a security program implementation which is close to real-time. A memory file can be added to the data filtering stage to save attack-type commands from earlier command scans.

After that step is completed, the system should be expanded from a two computer mini-network to a typical large scale network of fifty or sixty machines. This would be a good investigation to see if the system scales well to large amounts of data. It would also be interesting to see if the data collected for the mini-network (command usage, attack signature detections, false positives) was a good sample of the activity present on the larger network.

To steer all future work in this area in the right direction, it would be beneficial for the Air Force and the rest of DOD for the Air Force Information Warfare Center to set up a dedicated research program at AFIT to explore the future of intrusion detection.

With a continual program centered on a studying the various aspects of computer security, graduate students can build on each other's work to produce tools and techniques for securing Air Force systems. As distributed systems become more widespread in the future, a faculty member with experience in securing these types of systems should be added to guide this research.

6.3 Closing Thoughts

Intrusion detection will become even more important as the world gets wired online. More people connected through heterogeneous systems make the odds of being attacked greater each day. Investment in intrusion detection systems and further research in this area will keep the network intruders off the front pages of the newspapers and prevent compromise of sensitive computing resources.

Appendix A. Attack Signatures

The purpose of this appendix is to present the attack signatures that the security program was designed to find. Each signature is explained and then followed by the method used to detect it. The source for this appendix is a fax from the Air Force Information Warfare Center written by Dr. E. Eugene Schultz and Marvin J. Christensen (25).

A.1 The rdist Attack

The *rdist* command is a remote file distribution program. It can be exploited using two different techniques. The first step for both techniques begins with the attacker issuing the *rdist* command. In the first technique, the attacker erases a file named *rdist** and creates a link to this file. In the second technique, the attacker changes the path to the file by moving directories.

```
1. rdist
2. rm rdist*
3. ln (target file) rdist*

or

1. rdist
2. mv (target directory) (some directory)
3. mv (another directory) (target directory)
```

Table A.1 **rdist Attack Signature.**

The detection program searches for the *rdist, rm, ln,* and *mv* commands. The last three commands will only be reported if they are preceded by the *rdist* command.

A.2 The crash Attack

The *crash* command is used to examine system images. An attacker can crash a live system and search the memory for passwords. It can be exploited using three different techniques. The first step of the three techniques is to issue the crash command. In the first technique, the attacker performs a shell escape and issues the *strings* command. In the second technique, the attacker issues the *rd* command. In the third technique, the

attacker uses the *od* command to dump a file into octal, decimal, hexadecimal, or ASCII form.

```
1. crash
2. !sh
3. strings /dev/mem

or

1. crash
2. rd

or

1. crash
2. od
```

Table A.2 **crash Attack Signature.**

The detection program searches for the *crash* and *strings* command. The *rd* and *od* command are not searched for, because the attacker must issue these while the *crash* command is running. The attacker must be within the crash process to issue *od* and *rd*, and this prevents these commands from appearing in the accounting log.

A.3 The loadmodule Attack

The *loadmodule* command is used by OpenWindows to install loadable drivers into the UNIX kernel. It is used by attackers to run a Trojan Horse program with privileges. After the Trojan Horse is created, environment variables must be changed and the *loadmodule* command issued.

```
1. Create Trojan Horse.
2. Modify environment variables.
3. loadmodule
```

Table A.3 **loadmodule Attack Signature.**

The detection program searches for *loadmodu* (*loadmodule* gets truncated to eight characters by the accounting log).

A.4 The expreserve Attack

The expreserve attack relies on an editor vulnerability which allows the attacker to link to any file. First an editor must be started with the commands *vi, view, vedit, ex, edit,* or *e*. Next the attacker performs a shell escape. The attacker switches to the target directory and links to the target file. Finally the attacker kills the editor process.

```
1. vi, view, vedit, ex, edit, or e
2. !sh
3. cd (target directory)
4. ln (target file) (target directory)
5. kill editor process
```

Table A.4 **expreserve Attack Signature.**

The detection program cannot detect this type of attack. The accounting log generated by lastcomm does not record *cd* (change directory) or *kill*. These commands are hidden under the *csh* command which does appear in the log. With only editor commands and the link command to search for, this attack signature would generate too many false positives. It was not included in the detection program.

A.5 The sync Attack

The *sync* command is used to force changed data blocks to disk. The *sync* account can be exploited by an attacker to obtain unauthorized privileges. This attack signature has three techniques. The first technique is to compile and link a Trojan Horse program. The second technique is to set an environment variable and then *su* to the *sync* account. The third technique is to set an environment variable and then *login* to the *sync* account.

The detection program cannot detect this type of attack. This attack relies heavily on using specific command line switches. This type of information is not recorded in the accounting logs. The commands *cc* and *ld* may be scanned for, but they occur quite frequently any time someone compiles a C language program. The *setenv* command is hidden by the *csh* command in the accounting log. This attack signature was not included in the detection program.

```
1.  cc -c -R -pic
2.  ld -assert pure-text

or

1.  setenv LD_
2.  su sync

or

1.  setenv LD_
2.  login -p sync
```

Table A.5 **sync Attack Signature.**

A.6 The trusted hosts Attack

The trusted hosts attack was part of the famous Robert T. Morris Internet worm (9). An attacker uses another's account to access a specific machine which the account trusts. Trust indicates the machine may be accessed without having to type in a password. There are two techniques. In the first technique, the attacker must first enter the *su* command to switch to a new user id. Then he issues the *rsh* command to execute a command on a remote system. The second technique is the same as the first, but without the *su* command.

```
1.  su (acct)
2.  rsh (target) (command)

or

1.  rsh (target) (command)
```

Table A.6 **trusted hosts Attack Signature.**

The detection program can find this attack by searching for the *su* and *rsh* commands. The trust environment is important for use of these commands. Some systems permit users to utilize these commands, but security experts recommend against free use of these commands (2).

A.7 The xkey Attack

The *xkey* attack can be used to steal keypresses from a shell window. The attacker must have access the X-server on the target machine. After achieving access, the *xkey* command is used.

```
1.  xkey (host):(number)
```

Table A.7 **xkey Attack Signature.**

The detection program can find this attack by searching for the *xkey* command. The best protection against this attack is probably through server refusal to access other computers' Xlib directories.

A.8 The mount Attack

This attack signature was mislabeled in my source as the anonymous ftp attack (25). I will call it the mount attack. "In this attack pattern, the attacker first pings hosts, then executes rpcinfo to determine which hosts are NFS servers. The attacker next executes showmount to discover which file partitions can be exported, and finally, mounts the desired file partition" (25).

```
1.  ping
2.  rpcinfo
3.  showmount
4.  mount
```

Table A.8 **mount Attack Signature.**

The detection program searches for the *mount* command. The other commands show up in the accounting log, but are not important enough to scan for. Schultz states that the first three commands can be executed weeks or months before the attack is completed with the mount command (25). Normal users will not use the *mount* command, so the detection program searches for only the last phase of this attack.

A.9 The brute force Attack

The brute force attack occurs when an attacker attempts to gain system access by guessing the appropriate username/password combinations.

```
1. ftp, telnet, or rlogin
2. Guess username/password
3. Repeat 1 and 2.
```

Table A.9 **brute force Attack Signature.**

The detection program does not look for this attack signature. Not enough information is available in the accounting logs (such as the success or failure of a given command). Other system administrator utilities report some types of brute force attacks (see Figure 2.1).

A.10 The ftp Attack

The attacker executes *mkdir, chmod,* or *mknod* to create directories or change permissions for use in a later attack.

```
1. ftp
2. mkdir, chmod, or mknod
```

Table A.10 **ftp Attack Signature.**

The detection program does not look for this attack signature. The *ftp* command can be found through scanning, but *mkdir, chmod,* and *mknod* are omitted from the accounting log because they are used during a *ftp* session.

The best defense against this attack is probably to disallow these commands inside of *ftp.* The researcher tried two different *ftp* implementations. One allowed none of these commands to be used. Another allowed only *mkdir* to be executed.

A.11 The loadkeys Attack

This attack is not mentioned by Schultz or any other reference. It was independently uncovered by the researcher. The *loadkeys* attack is a denial of service attack. The attacker accesses a target system and redefines the keyboard. If an attacker can redefine the keyboard to all Z's, then the user has no choice but to reboot the computer. Usually *dumpkeys* is used first to capture the current keyboard definitions to a file. Then the file is maliciously edited and loaded back to the target computer.

```
1.  dumpkeys (file)
2.  loadkeys (file)

or

1. loadkeys (file)
```

Table A.11 **loadkeys Attack Signature.**

The detection program searches for the *dumpkeys* and *loadkeys* commands.

A.12 Summary

Seven of the eleven attack signatures may be detected through the use of the accounting logs. It is important for a system administrator to understand which attacks can be detected and which can go unnoticed.

Bibliography

1. Baran, Fuat, et al. *Security Breaches: Five Recent Incidents at Columbia University.* Tech Report., Center for Computing Activites: Columbia University, 1990.
2. Brand, Russel L. *Coping with the Threat of Computer Security Incidents - A Primer from Prevention through Recovery.* Technical Report, June 8 1990.
3. Brewin, Bob. "DOD to brief White House on hacker attacks," *Federal Computer Week* (25 July 1994).
4. Brewin, Bob and Elizabeth Sikorovsky. "Hackers storm DOD nets," *Federal Computer Week* (11 July 1994).
5. Cheswick, Bill. *An Evening with Berferd in Which a Cracker is Lured, Endured, and Studied.* Technical Report, AT&T Bell Labratories.
6. Cheswick, William R. and Steven M. Bellovin. *Firewalls and Internet Security - Repelling the Wily Hacker.* Addison-Wesley Publishing Company, 1994.
7. Curry, David. *Improving the security of your UNIX system.* Technical Report, SRI International, April 1990.
8. Curry, David. *UNIX System Security: a guide for users and system administrators.* Reading, MA: Addison-Wesley Publishing Company, Inc., 1992.
9. Eichin, Mark W. and Jon A. Rochlis. *With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988.* Technical Report, Cambridge: Massachusetts Institute of Technology, 1989.
10. Farrow, Rik. *UNIX System Security - How to Protect Your Data and Prevent Intruders.* Addison-Wesley Publishing Company, Inc., 1991.
11. Ferbrache, David and Gavin Shearer. *UNIX Installation - Security and Integrity.* PTR Prentice Hall, 1993.
12. Garfinkel, Simson and Gene Spafford. *Practical UNIX Security.* Sebastopol, CA: O'Reilly and Associates, Inc., 1991.
13. Garvey, Thomas D. and Teresa F. Lunt. "Model-Based Intrusion Detection." *14th National Computer Security Conference.* October 1-4 1991.
14. Giarratano, Joseph and Gary Riley. *Expert Systems - Principles and Programming.* PWS Publishing Company, 1994.
15. Hubley, Mary and Gary Brader. *Securing UNIX Systems.* Technical Report, Datapro Reports on Information Security, January 1994.
16. Kumar, Sandeep and Eugene H. Spafford. *An Application of Pattern Matching in Intrusion Detection.* Technical Report, Purdue University, June 17 1994.
17. Lankewicz, Linda and Mark Benard. "Real-time Anomaly Detection Using a Nonparametric Pattern Recognition Approach." *Computer Security Applications Conference.* December 1991.

18. Lister, Justin, "Intrusion Detection System Review." Post to the Intrusion Detection Mailing List, August 1994.

19. Lunt, Teresa, "Intrusion Detection Statistics." Electronic Mail Message, September 1994.

20. Lunt, Teresa, "NIDES." Post to the Intrusion Detection Mailing List., 1994.

21. Lunt, Teresa F. "Automated Audit Trail Analysis and Intrusion Detection: A Survey." *Computer Security Conference Proceedings*. 1988.

22. Marshall, Victor H. *Intrusion Detection in Computers*. Technical Report, Booz, Allen, and Hamilton Inc., January 1991.

23. McCarron, John. "Applications of Knowledge Based Systems Techniques to Detect Computer Systems Intrusions." *13th National Computer Security Conference*. October 1-4 1990.

24. Roth, Kristina, "Thesis Feedback." Electronic Mail Message, October 1994.

25. Schultz, E. Eugene and Marvin J. Christensen. "Distributive Intrusion Detection System (DIDS), Petri Net Diagrams, Deliverable 2.1.b.3." March 1994.

26. Schultz, E. Eugene, et al. "Computer Emergency Response Teams: Lessons Learned." *13th National Computer Security Conference*. October 1-4 1990.

27. Sebring, Michael M., et al. "Expert Systems in Intrusion Detection: A Case Study." *11th Computer Security Conference Proceedings*. 1988.

28. Smaha, Steve, "Using Non-Audit Data for Misuse Detection." 14th Intrusion Detection Workshop, November 1994.

29. Snapp, S.R., et al. "Detecting Intrusions Through Attack Signature Analysis." Lawerence Livermore Laboratory, 11 October 1991.

30. Snapp, Steven R., et al. "DIDS (Distributive Intrusion Detection System) - Motivation, Architecture, and An Early Prototype." *14th National Computer Security Conference*. October 1991.

31. Southerton, Alan and Jr. Edwin C. Perkins. *The UNIX and X Command Compendium - A Dictionary for High Level Computing*. John Wiley & Sons, Inc., 1994.

Vita

Lieutenant David Robert Landry was born in Wiesbaden, Germany on 3 May 1971. He received the IEEE Computer Society second place award at the 40th International Science and Engineering Fair in 1989. In 1993, he graduated from the United States Air Force Academy with his undergraduate degree in Computer Science. Following graduation from the Air Force Institute of Technology, Lieutenant Landry will work as a member of the Software Engineering Division of the Air Force Wargaming Institute at Maxwell Air Force Base, Alabama.

Permanent address: 45887 Cabin Branch Drive
Sterling, VA 20164